Agenda

- **Introduction**
- **Configuring and Navigating**
- **Parameterized routes**
- **Nested (or) Child Routes**

## Introduction

Angular provides '@angular/router' library to enable routing in our application.

Routing is used to navigate from one view to another when user performs any task.

Angular router provides us to pass **optional parameters** along to the corresponding component to display the specific content.

We can bind router to the links that will navigate us to corresponding view when user clicks.

**RouterLink**, **RouterLinkActive** and **RouterOutlet** are directives provided by the Angular **RouterModule** package. They are readily available for you to use in the template.

**Router Links:**

```
<style>
 .red{
   background-color:red
 }
</style>
<a routerLink="path" routerLinkActive="red">first route</a>
OR
<button class="btn btn-primary" routerLink="path" routerLinkActive="red">first route</button>
```

The **routerLink** attribute is mostly used either on **<a> or <button>** tags which gives the router control over the element. And **routerLinkActive** attributes will select the route as default route.

**Router Outlet:**

This directive displays the inner content of the corresponding route component template in the current view.

```
<router-outlet></router-outlet>
```

**Router Module:**

This is an angular **NgModule** that provides the necessary services and directives which helps us to navigate between views.

**App.routing.ts**

```
import { ModuleWithProviders } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
const routes: Routes = [
    { path: 'first-route', component: FirstComponent },
];
export const routing: ModuleWithProviders = RouterModule.forRoot(routes);
```

There are two ways to register our routes:

**RouterModule.forRoot**: **forRoot** creates a module that contains all the directives, the given routes, and the router service itself.

**RouterModule.forChild**: **forChild** creates a module that contains all the directives and the given routes, but does not include the router service.

## Configuring and Navigating

Now let us create an application.

**Step1: Creating models**

Let us create models that are required for our application

File: **Department.ts**

```
export class Department {
    DeptId: number;
    DeptName: string;
}
```

File: **employe.ts**

```
export class Employee {
    EmpId: number;
    EmpName: string;
    EmpSalary: number;
    Department: Department;
}
```

**Step2: Create the components required**

First let us create child components **EmpComponent** and **DeptComponent**

File: **employee.component.ts**

```
import { Component} from '@angular/core';
import { Employee } from './app.models';

@Component({
    templateUrl: './Employee.html'
})
export class EmployeeComponent{
    emps: Employee[];
```

```
  constructor() {
    var lstEmp: Employee[] = [
      { EmpId: 1, EmpName: "Phani", EmpSalary: 15000, Department: { DeptId: 1, DeptName: "D1" } },
      { EmpId: 2, EmpName: "Kranth", EmpSalary: 115000, Department: { DeptId: 2, DeptName: "D2" } }
    ];
    this.emps = lstEmp;
    }
}
```

In this application, I've created few static employees, if we want to make them dynamic we need to use **HTTP** services and fetch the data from server, we will discuss how to fetch data from server in services session.

File: **employee.html**

```
<table class="table table-bordered">
  <tr>
    <th>Name</th>
    <th>Department</th>
  </tr>
  <tbody>
    <tr *ngFor="let emp of emps">
      <td>{{emp.EmpName}}</td>
      <td>{{emp.Department.DeptName}}</td>
     </tr>
  </tbody>
</table>
```

Now create **DepartmentComponent**

File: **department.component.ts**

```
import { Component} from '@angular/core';
import { Department } from './department;

@Component({
  templateUrl:'./department.html'
})
export class DepartmentComponent{
  depts: Department[];
    constructor() {
    var lstDept: Department[] = [
      { DeptId: 1, DeptName: "D1" },
         { DeptId: 2, DeptName: "D2" }
       ];
       this.depts = lstDept;
    }
}
```

File: **department.html**

```
<table class="table table-bordered">
   <tr>
      <th>Name</th>
   </tr>
   <tbody>
      <tr *ngFor="let dept of depts">
         <td>{{dept.DeptName}}</td>
       </tr>
   </tbody>
</table>
```

Now let’s edit root component

File: **app.component.ts**

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';

@Component({
   selector: 'my-app',
   templateUrl: './my-app.html'
})
export class AppComponent {
     constructor(private router:Router) { }
    //Here instead of using automatic routing we can use manual routing using Router.Navigate.
    goHome() {
      this.router.navigate(['emp']);
   }
}
```

File: **my-app.html**

```
<div class="container">
   <h1>Office Management</h1>
   <nav>
```

```
        <button class="btn btn-warning" routerLink=" ">Go Home </button>
        <button class="btn btn-warning" routerLink="emp">Employees</button>
         <button class="btn btn-warning" routerLink="dept">Departments</button>
         <button class="btn btn-warning" (click)="goHome()">Departments</button>
    </nav>
    <br />
    <router-outlet></router-outlet>
</div>
```

**Step3: Create routing required**

File: **app.routing.ts**

```
import { ModuleWithProviders } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { EmployeeComponent } from './employee.component';
import { DepartmentComponent } from './department.component';

const routes: Routes = [
    { path: '', redirectTo: 'emp', pathMatch: 'full' },
    { path: 'emp', component: EmployeeComponent },
    { path: 'dept', component: DepartmentComponent },
    { path: **, component: FileNotFoundComponent },
 ];
export const routing: ModuleWithProviders = RouterModule.forRoot(routes);
```

**Note:** For the special case of an *empty* URL we also need to add the pathMatch: 'full' property so Angular knows it should be matching exactly the empty string and not *partially* the empty string.

The other possible **pathMatch** value is **'prefix'** which tells the router to match the redirect route when the remaining URL begins with the redirect route's prefix path.

**CATCH ALL ROUTE:** We can also add a *catch all* route by using the path **, if the URL doesn't match *any* of the other routes it will match this route. In our example above we are just showing the **FileNotFoundComponent.**

**File: filenotfound.component.ts**

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';

@Component({
    selector: 'my-app',
    template: `Request File/URL is not found`
})
export class FileNotFoundComponent {
}
```

**Step4: Setup NgModule & bootstrap the application**

File: **app.module.ts**

```
import { routing } from './app.routings';
import { EmployeeComponent } from './employee.component';
import { DepartmentComponent } from './department.component';
import { FileNotFoundComponent } from './filenotfound.component';

@NgModule({
   imports: [BrowserModule, routing],
   declarations: [AppComponent, EmployeeComponent, DepartmentComponent, FileNotFoundComponent],
  bootstrap: [AppComponent]
})
export class AppModule {  }
```

Refactor the routing configuration into a *routing module*

File: App-routing.ts

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

import { EmployeeComponent } from './employee.component';
import { DepartmentComponent } from './department.component';
import { FileNotFoundComponent } from './filenotfound.component';

const appRoutes: Routes = [
   { path: '', redirectTo: 'emp', pathMatch: 'full' },
   { path: 'emp', component: EmployeeComponent },
   { path: 'dept', component: DepartmentComponent },
   { path: **, component: FileNotFoundComponent },
];

@NgModule({
   imports: [
      RouterModule.forRoot(appRoutes, { enableTracing: true })
                        //enable Tracing is for debugging purposes only
```

```
  ],
  exports: [
    RouterModule
  ]
})
export class AppRoutingModule { }
```

Now you need to include **AppRoutingMoudle** in AppModule imports array.

```
import { AppRoutingModule } from './app-routing.module'
@NgModule({
  imports: [. . .,AppRoutingModule],
})
```

## Parameterized Routes

From the above example if we want to get the details of individual record of any Employee then we have to show the url as

Emp/1

Emp/2

. . . .

To achieve this we need to create one more route to accept the parameters.

File: **app.routings.ts**

```
const routes: Routes = [
  { path: '', redirectTo: 'emp', pathMatch: 'full' },
  { path: 'emp', component: EmpComponent },
  { path: 'emp/:id', component: EmpComponent },
  { path: 'dept', component: DeptComponent },
  { path: 'dept/:id', component: DeptComponent }
];
```

Now to accept the parameters in the target component we need to import **ActivatedRoute** class and inject it into our component**.** Target component fires when the user clicks on a link/button, then we need to fetch the parameter using **ActivatedRoute**

The parameters are wrapped in an Observable that will push the current route parameter value whenever the parameter is updated. We subscribe for any changes. When a new value is received we set the value to a property on our template. We could just as easily take this value as an ID to retrieve some data from a API. We capture the subscription in a property so when the component is destroyed we unsubscribe preventing any memory leaks.

File: **employee.component.ts**

Modify the constructor as follows

```
import { Component } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { Employee } from './Employee';

@Component({
  templateUrl: './Employee.html'
})
export class EmployeeComponent {
  emps: Employee[];
  selectedEmployee: Employee = null;
  constructor(private route: ActivatedRoute) {
    var lstEmp: Employee[] = [
      { EmpId: 1, EmpName: "Phani", EmpSalary: 15000, Department: { DeptId: 1, DeptName: "D1" } },
      { EmpId: 2, EmpName: "Kranth", EmpSalary: 115000, Department: { DeptId: 2, DeptName: "D2" } }
    ];
    this.emps = lstEmp;
  }
 paramsSub: any;
 ngOnInit()
 {
     this.paramsSub = this.route.params.subscribe(params => {
          if (params["id"] != null)
                      this.selectedEmployee = this.emps.filter(e => e.EmpId == params["id"])[0];
          });
 }
 ngOnDestroy()
 {
      this.paramsSub.unsubscribe();
 }
}
```

File: **employee.html**

Add one more column in the table as follows

```
<table class="table table-bordered" style="width:50%">
  <tr>
    <th>Name</th>
    <th>Department</th>
```

```
        <th></th>
        <th></th>
    </tr>
    <tbody>
        <tr *ngFor="let emp of emps">
            <td>{{emp.EmpName}}</td>
            <td>{{emp.Department.DeptName}}</td>
            <td><a [routerLink]="['/emp',emp.EmpId]">Details</a></td>
            <td><a [routerLink]="['/emp',{id:emp.EmpId, foo:'foo'}]">Details</a></td>
        </tr>
    </tbody>
</table>
<hr />
<div *ngIf="selectedEmployee!=null">
  Selected Employee Details:
  <hr />
  Name: {{selectedEmployee.EmpName}} <br />
  Salary: {{selectedEmployee.EmpSalary}} <br />
  Department {{selectedEmployee.Department.DeptName}}
</div>
```

## Nested (or) Child Routes

To view **route within other** route we use nested (or) child routes i.e. now we will have two <router-outlet> tags, where one route will be primary and other one will be child of the primary.

Let us modify the above example as follows, create a new folder **NestedRoutes** and create the files within.

File: **header.component.ts**

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';

@Component({
    selector: 'app-head',
    templateUrl: './head.html'
})
export class HeadComponent {
}
```

File: **head.html**

```
<div class="clearfix">&nbsp;</div>
  <nav class="navbar navbar-light" style="background-color:rgba(211, 204, 204, 0.30)">
    <a class="navbar-brand" routerLink="home">Home</a>
    <ul class="nav navbar-nav">
      <li class="nav-item">
        <a class="nav-link" routerLink="emp">Employees</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" routerLink="dept">Departments</a>
      </li>
    </ul>
  </nav>
```

File: **home.component.ts**

```
import { Component } from '@angular/core';
@Component({
  templateUrl: './home.html'
})
export class HomeComponent {
}
```

File: **home.html**

```
<div class="well">
  <h1>Welcome (this is home)</h1>
</div>
<div>
  <router-outlet></router-outlet>
</div>
```

Modify the **AppComponent** as follows

File: **app.component.ts**

```
import { Component } from '@angular/core';
@Component({
  selector: 'my-app',
  templateUrl: './my-app.html'
})
export class AppComponent {}
```

Modify **template.html** as follows

File: **template.html**

```
<div class="clearfix">&nbsp;</div>
<div class="container">
  <app-head></app-head>
  <div>
    <router-outlet></router-outlet>
  </div>
</div>
```

Here we have child <router-outlet>

Now **routing** is the main thing to achieve **child/nested routes**

**Note:** Path which has **redirectTo** property cannot have **children**

File: **app.routings.ts**

```
import { ModuleWithProviders } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { AppComponent } from './app.component';
import { HomeComponent } from './home.component';

const routes: Routes = [
  {
    path: '', component: HomeComponent,
    children: [
      { path: '', redirectTo: 'emp', pathMatch: 'full' },
      { path: 'emp', component: EmployeeComponent },
      { path: 'emp/:id', component: EmployeeComponent },
      { path: 'dept', component: DepartmentComponent },
      { path: 'dept/:id', component: DepartmentComponent }
    ]
  },
  { path: 'home', component: HomeComponent }
];
export const routing: ModuleWithProviders = RouterModule.forRoot(routes);
```

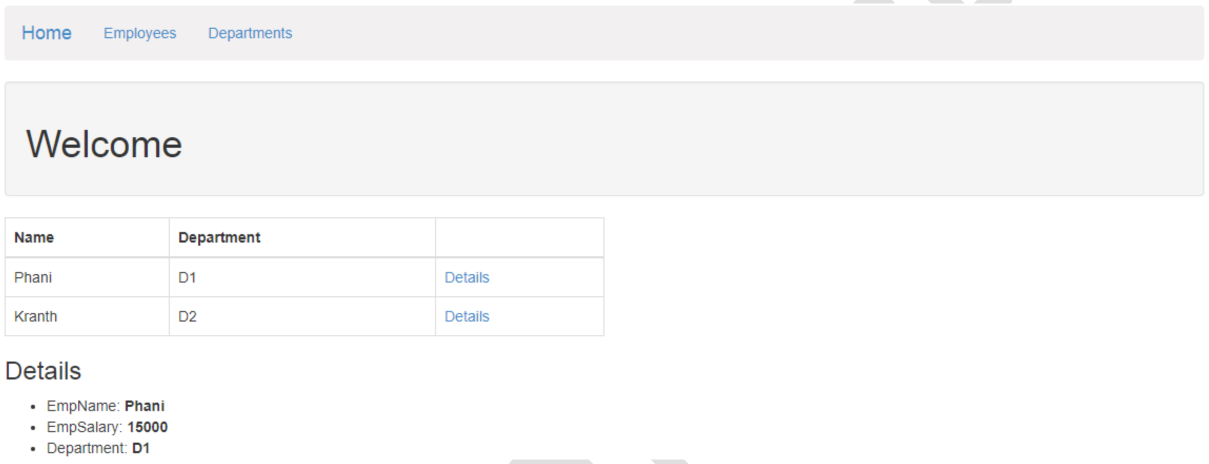
1. ***redirectTo*** and ***children*** cannot be used together.
2. Here our default route i.e. route with **empty path** will render **HomeComponent**

3. In **HomeComponent** we have primary <router-outlet>
4. Now the route with empty path is rendering **HomeComponent**, now we have created **child routes** for this route in which again its having **one route with empty path** this will redirect us to **EmpComponent.** So our default output will be list of employees

Don't forget to declare all the components in root **NgModel**

```
declarations: [AppComponent, EmpComponent, DeptComponent, HomeComponent, HeadComponent],
```

**Output:** Now run the application.



Welcome

| Name | Department | |
|---|---|---|
| Phani | D1 | Details |
| Kranth | D2 | Details |

Details

- EmpName: **Phani**
- EmpSalary: **15000**
- Department: **D1**